PASSED

Group 3 - lab 2

Niclas Lovsjö & Maxime Bonneau

12 february 2016

Assignment 1

1.

We start by dividing the data in training and test datasets.

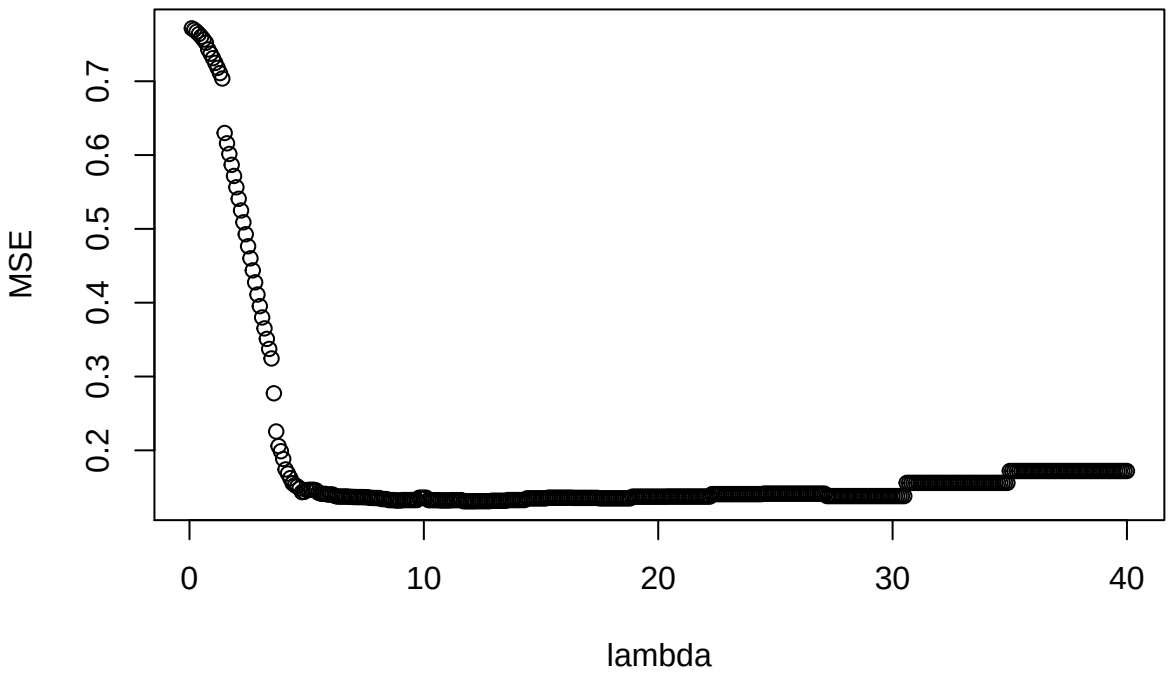
2.

Following is our function myMSE, which computes the MSE for a loess regression:

```
myMSE <- function(lambda, pars, printer = TRUE){
  X <- pars[[1]]
  Y <- pars[[2]]
  Xtest <- pars[[3]]
  Ytest <- pars[[4]]
  reg <- loess(Y~X, data=train, enp.target = lambda)
  fit_reg <- predict(reg, newdata = Xtest)
  MSE <- 1/nrow(test)*sum((fit_reg-Ytest)^2)
  if(printer == TRUE){
    print(MSE)
  }
  return(MSE)
}
```

3.

The plot under is the result of the compuation of myMSE function for lambda between 0.1 and 40.



We have the result that the best MSE is for a lambda = 11.7, 11.8, 11.9, 12, 12.1, 12.2, where the value of myMSE is: 0.131047. 400 evaluations of myMSE were needed.

4.

We try now to find the minimum value of myMSE by usin optimize function. We get the following result:

```
## [1] 0.1358018
## [1] 0.1412809
## [1] 0.1326581
## [1] 0.1401702
## [1] 0.1325391
## [1] 0.1321441
## [1] 0.1325542
## [1] 0.1321441
## [1] 0.1325391
## [1] 0.1321441
## [1] 0.1321441
## [1] 0.1321441
## [1] 0.1321441
## [1] 0.1321441
## [1] 0.1321441
## [1] 0.1321441
## [1] 0.1321441
## [1] 0.1321441
```

```
## $minimum
## [1] 10.69361
##
## $objective
## [1] 0.1321441
```

So the result is not the same than the result we had in the previous question, but the value of myMSE is a little bit lower, so this result seems to be better. This time, 18 evaluations of myMSE were needed, which is also faster.

5.

Finally we use the method BFGS of the optim function to try to find the minimum of myMSE. We get the following result:

```
## [1] 0.1719996
## [1] 0.1719996
## [1] 0.1719996

## $par
## [1] 35
##
## $value
## [1] 0.1719996
##
## $counts
## function gradient
##        1        1
##
## $convergence
## [1] 0
##
## $message
## NULL
```

It is different to the two previous methods, with a higher value of min(myMSE). Thanks to the counts, we can see that myMSE was called only once. But with the printings of myMSE, it looks like it has been called three times. So this method is much faster, but less accurate than the two previous ones. This is probably due to the fact that there are a lot of equal values of MSE for high values of lambda, so the optim function thinks that myMSE is constant.

To conclude, we do no get a good result with this method, but with the two previous ones, we can see that if we want an accurate result, we need to evaluate the function to minimize some times more than with optim. In our opinion, optimize is a quite fast function for a good result, so this is the one to favour.

Assignment 2

We load in some data X_i, $i \in [1, ..., 100]$ distributed as $N(\mu, \sigma)$. We're looking for the log-likelihood of the density $f_{\mu,\sigma}(X_i)$. We have that,

$$
\begin{aligned}
l = \log\mathcal{L}(\mu, \sigma|X) &= \log\left(\prod_i^n P(X|\mu,\sigma)\right) \\
&= \log\left(\left(\frac{1}{2\pi\sigma^2}\right)^{n/2}\exp\left(-\frac{\sum_i^n (X_i-\mu)^2}{2\sigma^2}\right)\right) \\
&= -\frac{n}{2}\log(\sigma^2) - \frac{n}{2}\log(2\pi) - \frac{\sum_i^n (X_i-\mu)^2}{2\sigma^2}.
\end{aligned}
$$

Maximizing the log-likelihood, i.e. taking the partial derivatives and setting to 0 and solving yields the following ML-estimators of μ and σ^2,

$$
\begin{aligned}
\frac{\partial l}{\partial \mu} &= \frac{\sum_i^n (X_i-\mu)}{\sigma^2} = 0 \\
&\Leftrightarrow \hat{\mu} = \frac{\sum_i^n X_i}{n} = \bar{X} \\
\frac{\partial l}{\partial \sigma} &= -\frac{n}{\sigma} + \frac{\sum_i^n (X_i-\mu)^2}{\sigma^3} = 0 \\
&\Leftrightarrow \hat{\sigma}^2 = \frac{1}{n}\sum_i^n (X_i-\bar{X})^2
\end{aligned}
$$

Using these estimators for our dataset yields,

```
x_bar
```

```
## [1] 1.275528
```

```
sigma_bar
```

```
## [1] 4.023942
```

We want to use the log-likelihood since its derivatives is easier to handle. This is possible since the log-function is monotonically increasing, i.e. if $x \leq y$ then $f(x) \leq f(y)$ for all x, y.

We also want to maximize the likelihood numerically, i.e. minimize the negative log-likelihood. Both using the *Conjugate gradient method* and the *BFGS*-method. We will do this both with and without setting the gradient-vector as specified in the derivation above. The code will output answers to whether the functions converged and how many iterations it used for calling the functions and gradients.

```
## CG did converge.
## Estimated parameters:
## mu = 1.275528 sigma = 4.023941 .
## Number of counts;
## Function: 178
## Gradient: 61
##
## BFGS did converge.
## Estimated parameters:
## mu = 1.275528 sigma = 4.023942 .
```

```
## Number of counts;
## Function: 31
## Gradient: 15
##
## CG_gr did converge.
## Estimated parameters:
## mu = 1.275528 sigma = 4.023942 .
## Number of counts;
## Function: 37
## Gradient: 13
##
## BFGS_gr did converge.
## Estimated parameters:
## mu = 1.275528 sigma = 4.023942 .
## Number of counts;
## Function: 36
## Gradient: 16
```

All functions are able to find the μ and σ-estimations. We see that the CG-method does benefit from getting an explicit gradient supplied, and we choose this as our best method in this case.

Appendix

```
## ---- echo = FALSE, message = FALSE----------------------------------------
mr <- read.csv2("C:/Users/Maxime/ENSAI/Liu/Computational statistics/Lab 2/mortality_rate.csv", header =
LMR <- log(mr$Rate)
mr <- data.frame(mr,LMR)

n <- dim(mr)[1]
set.seed(123456)
id <- sample(1:n, floor(n*0.5))
train <- mr[id,]
test <- mr[-id,]

## --------------------------------------------------------------------------
myMSE <- function(lambda, pars, printer = TRUE){
  X <- pars[[1]]
  Y <- pars[[2]]
  Xtest <- pars[[3]]
  Ytest <- pars[[4]]
  reg <- loess(Y~X, data=train, enp.target = lambda)
  fit_reg <- predict(reg, newdata = Xtest)
  MSE <- 1/nrow(test)*sum((fit_reg-Ytest)^2)
  if(printer == TRUE){
    print(MSE)
  }
  return(MSE)
}

## ---- echo = FALSE---------------------------------------------------------
MSE <- rep(0,nrow(test))
x <- seq(0.1,40,0.1)
```

```r
for(i in 1:400){
  MSE[i] <- myMSE(lambda = i*0.1, pars = list(train$Day, train$LMR, test$Day, test$LMR), printer = FALSE
}
plot(x, MSE, xlab = "lambda")

## ---- echo = FALSE-------------------------------------------------------
optimize(function(lambda) myMSE(lambda, list(train$Day, train$LMR, test$Day, test$LMR)), interval = c(0

## ---- echo = FALSE-------------------------------------------------------
f <- function(lambda){
  return(myMSE(lambda, list(train$Day, train$LMR, test$Day, test$LMR)))
}
optim(35, f, method = "BFGS")

## ---- echo = FALSE-------------------------------------------------------
load("data.RData")
n<-length(data)

## ---- echo = FALSE-------------------------------------------------------
x_bar<-sum(data)/n
sigma_bar<-sum((data-x_bar)^2)/n

## ------------------------------------------------------------------------
x_bar
sigma_bar

## ---- echo = FALSE-------------------------------------------------------
loglik<-function(theta,x){
  loglik<-(-n/2*log(theta[2])-n/2*log(2*pi)-sum((x-theta[1])^2)/(2*theta[2]))
  return(-loglik)
}
theta<-list(mu=0,sigma=1)
gradient<-function(theta,x){
  return(-c(sum(x-theta[1])/theta[2],
              -n+sum((x-theta[1])^2)/theta[2]))
}

## ----include=FALSE, echo = FALSE-----------------------------------------
optim_funs<-list()
cg<-optim(par=theta,fn=loglik,method="CG",x=data)
optim_funs[[1]]<-cg
bfgs<-optim(par=theta,fn=loglik,method="BFGS",x=data)
optim_funs[[2]]<-bfgs
#note that we get NA's because sigma takes on neg.values.
#to overcome this, we could set the params to exp and then take the log afterwards.
cg_gr<-optim(par=theta,fn=loglik,gr=gradient,method="CG",x=data)
optim_funs[[3]]<-cg_gr
bfgs_gr<-optim(par=theta,fn=loglik,gr=gradient,method="BFGS",x=data)
optim_funs[[4]]<-bfgs_gr

## ---- echo = FALSE-------------------------------------------------------
names<-c("CG","BFGS","CG_gr","BFGS_gr")
for (i in 1:4){
```

```
  if(optim_funs[[i]]$convergence==0){
    cat(names[i],"did converge.","\n")
    cat("Estimated parameters:","\n")
    cat("mu =",optim_funs[[i]]$par[1],"sigma =",optim_funs[[i]]$par[2],".\n")
    cat("Number of counts;","\n")
    cat("Function:",optim_funs[[i]]$counts[1],"\n")
    cat("Gradient:",optim_funs[[i]]$counts[2],"\n")
    cat("\n")}

  else {print(optim_funs[i],"did not converge")}
}

## ----code=readLines(knitr::purl("C:/Users/Maxime/ENSAI/Liu/Computational statistics/Lab 2/Group 3 - la
##
```